Hybrid Sensor Fusion Framework for Perception in Autonomous Vehicles

Babak Shahian Jahromi
University of Illinois at Chicago
`bshahi2@uic.com`

Theja Tulabandhula
University of Illinois at Chicago
`theja@uic.edu`

Sabri Cetin
University of Illinois at Chicago
`scetin@uic.com`

Abstract

There are many sensor fusion frameworks proposed in the literature using different sensors and methods configurations. Most focus has been on improving the accuracy performance; the implementation feasibility of these frameworks in an autonomous vehicle is less explored. Some fusion architectures can perform very well in lab conditions using powerful computational resources; however, in real-world applications, they cannot be implemented in an embedded edge computer due to their high computational need. We propose a new hybrid multi-sensor fusion pipeline configuration that performs environment perception for autonomous vehicles such as road segmentation, obstacle detection, and tracking. This fusion framework uses an improved encoder-decoder based Fully Convolutional Neural Networks (FCN) and a traditional Extended Kalman Filter (EKF) nonlinear state estimator methods. It also uses a configuration of camera, lidar, and radar sensors that are best suited for each fusion method. The goal of this hybrid framework is to provide a relatively lightweight, modular, and robust fusion system solution. It uses modified algorithms that improve environment perception accuracy and real-time efficiency compared to benchmark models that can be used in an autonomous vehicle embedded computer. Our fusion algorithm shows better performance in various environment scenarios compared to baseline techniques. Moreover, the algorithm is implemented in a vehicle and tested using actual sensor data collected from a vehicle, performing real-time environment perception.

1 Introduction

There is numerous research on the environment perception for autonomous vehicles, including the sensors used in an AV, sensor data processing, and various fusion algorithms. The sensors can be categorized into three main categories: (1) camera, (2) lidar, and (3) radar. Also, the fusion algorithms can be categorized into two main categories: (1) sensor fusion using state estimators, i.e., Bayesian filters (BF), and (2) machine learning-based methods, i.e., deep neural networks (DNN). Literature focus has been often on improving the algorithm accuracy performance; the implementation feasibility of these algorithms in an autonomous vehicle, however, has been less explored. The need for an efficient, lightweight, modular, and robust pipeline is essential. Therefore, a sensor fusion method configuration that balances a trade-off between fusion model complexity and real-world real-time applicability while improving the environment perception accuracy is fundamental.

In this research paper, we propose a hybrid sensor fusion framework configuration for autonomous driving. In addition to accuracy improvement, this modular framework takes into account and

Machine Learning for Autonomous Driving Workshop at the 33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada.

combines the strengths of nonlinear state estimators, i.e., Extended Kalman Filter technique with the strength of deep learning algorithms, i.e., Fully Convolutional Network, based on the sensor type and configuration. This hybrid sensor fusion technique is used for understanding the environment around the autonomous vehicle to provide rich information on the safe driveable road region as well as detecting and tracking the objects on the road. This method was tested via an embedded in-vehicle computer, and the results were compared to ground truth information.

2 Hybrid sensor fusion algorithm overview

The three main perception sensors used in autonomous vehicles have their strengths and weaknesses; therefore, the information from them needs to be combined (fused) to make the best sense of the environment. Here, we propose and implement a hybrid sensor fusion algorithm framework to this end. The hybrid sensor fusion algorithm consists of two parts that run in parallel, as shown in fig. 1. In each part, a set configuration of sensors and a fusion method is used that is best suited for the fusion task at hand.

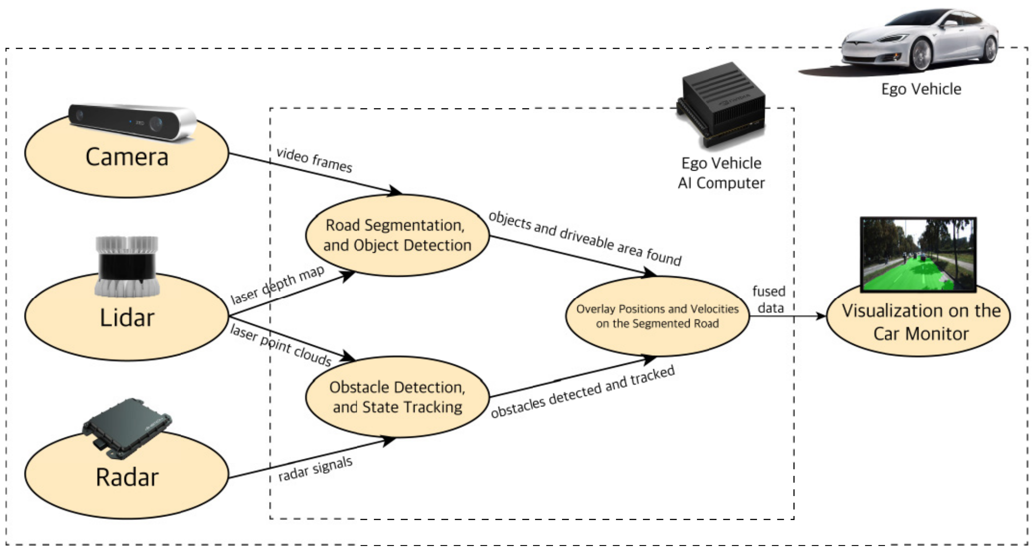
Ego Vehicle
Camera
video frames
Ego Vehicle
AI Computer
Road Segmentation, and Object Detection
objects and driveable area found
Lidar
laser depth map
laser point clouds
Overlay Positions and Velocities on the Segmented Road
fused data
Visualization on the Car Monitor
Obstacle Detection, and State Tracking
obstacles detected and tracked
Radar
radar signals

Figure 1: Multi-sensor fusion algorithm pipeline.

The first part deals with high-resolution tasks of object classification, localization, and semantic road segmentation by using the camera (vision) and lidar sensors. The camera's raw video frames and the lidar's depth channel are combined before being sent to our proposed Fully Convolutional Networks (FCNx) in charge of the object classification and road segmentation tasks. The combination of camera and lidar with FCNx architecture gives us the best sensor-method combination for performing classification and segmentation. In this paper, we prioritize real-time performance and accuracy. We perform segmentation for two classes: the free space (driveable area) of the road and not driveable area of the road for our autonomous vehicle. We compare our free navigation space detection architecture with baseline benchmarks. For automated driving, knowledge of driveable space, and obstacle classes on the road is essential for path planning and decision making.

The second part deals with the task of detecting objects and tracking their states by using lidar and radar sensors. The lidar point cloud data (PCD) and radar signals are processed and fused at the object level. The lidar and radar data processing will result in clusters of obstacles on the road within the region of interest (ROI) with their states. The fusion of processed sensor data at the object level is referred to as late fusion. The resulting late fused lidar and radar data is sent to a nonlinear state estimation method to best combine the noisy measured states of each sensor. Since the motion of obstacles like cars and sensor measurement models can be nonlinear; we use a classic Extended Kalman Filter (EKF). Knowing and tracking the states of the obstacles on the road helps to predict and account for their behavior in the AV path planning and decision making stacks. Finally, we overlay each fusion output and visualize them on the car monitor. The tracking using EKF is implemented so we can show and test the complete implementation of this framework in real-world scenarios with actual sensor data.

3 Object classification and road segmentation using deep learning

For road semantic segmentation, we use camera images as well as depth information from the lidar. We combine these raw data at the depth channel, which results in an RGBD image with a depth channel of size four. We then send this information to our proposed Fully Convolutional Network (FCNx) architecture. FCNx is based on an encoder-decoder architecture and consists of two parts: an encoder to extract the features from the RGBD image at various levels from high level to low level. Also, a decoder that will upsample the features, so we get detailed segmented road images back showing the free navigation space. The model is trained and tested using NVIDIA GPUs. Finally, the model performance is evaluated by comparing its predictions with the ground truth labels and to segmentation architecture benchmarks.

3.1 Camera-lidar raw data fusion

The camera and lidar sensors provide the input data for the FCNx. The camera provides 2-dimensional (2D) color images of three RGB channels. The lidar provides a high-resolution depth map in addition to the point cloud data. We combine the unprocessed raw data of lidar and camera (early fusion). The resulting RGBD image will have four channels. The RGBD image contains the 2D appearance features of the camera image with 3D depth features of lidar to give us a rich illumination-invariant spatial image. This image is fed to the FCNx to extract road features for semantic segmentation and a CNN for object detection and localization depending on the application.

3.2 Proposed fully convolutional network (FCNx) architecture

The architecture consists of two main parts: an encoder and a decoder. First, we have the encoder; it is based on the VGG16 architecture [1], with its fully connected layers replaced with convolutional layers. The purpose of the encoder is to extract features and spatial information from our four-channel RGBD input image. It uses a series of convolutional layers to extract features and max-pooling layers to reduce the size of the feature maps. Second, we have a decoder, which is based on the FCN8 architecture [2]. Its purpose is to rebuild the prediction of pixel classes back to the original image size while maintaining low-level and high-level information integrity. It uses transposed convolutional layers to upsample the output of the last convolutional layer of the encoder [3]. Also, it uses skip convolutional layers to combine the finer low-level features from encoder layers with the coarser high-level features of transposed convolutional (upsampled) layers.

In our proposed network, which builds on the above shown in fig. 2, we combine the encoder output from layer four with the upsampled layer seven. This combination is an element-wise addition. This combined feature map is then added to the output of the third layer skip connection. In the skip connection of the layer three output, we utilize a second convolutional layer to further extract features from layer three output. The addition of this convolutional layer adds some features that would be extracted in layer four. Including some basic layer four level feature maps will help the layer three skip connection to represent a combined feature map of layer three and layer four. This combined feature map is then added to the upsampled layer nine, which itself represents a combined feature map of layer seven and layer four. This addition is shown to give a better accuracy and lower Cross-Entropy loss compared to the base VGG16-FCN8 architecture. The better performance can be explained by the fact that, having some similar layer four feature maps, can help better align the extracted features when performing the last addition. We will refer to our proposed architecture as FCNx. The purpose of the FCNx applied to camera and lidar fused raw data is to segment the road image into driveable free space area and non-driveable area. The output can be further processed and sent through a plug and play detector network (YOLO [4, 5, 6], SSD [7]) for object detection and localization depending on the situation.

3.3 Network experiments procedure

For our FCNx architecture, we use a combined dataset for training. The dataset is a combination of UC Berkeley DeepDrive (BDD) dataset [8], University of Toronto KITTI dataset [9] and a self-generated dataset generated from our sensors installed in our test vehicle summing up to more than 3000 images. Our sensor data was acquired from a ZED camera and an Ouster lidar mounted on our test vehicle capturing data from streets of the city of Chicago. The BDD and KITTI dataset are

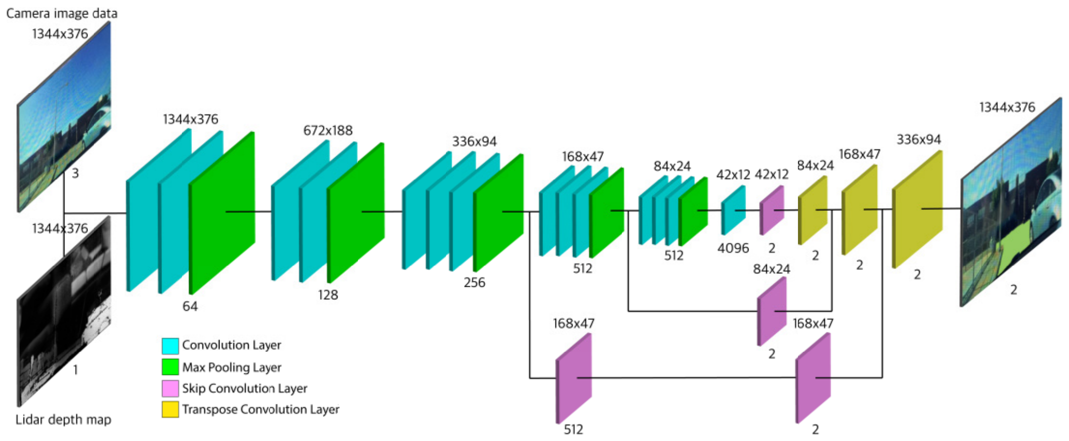

Figure 2: Overview of the object detection and road segmentation architecture via a Fully Convolutional Neural Network. The tiles and colors indicate different layers of the FCNx and their type respectively. Encoder consists of: convolution (teal) and max pooling (green) layers. Decoder consists of: skip convolution (purple) and transposed convolution (yellow). The size and number of feature maps are shown at the top and bottom of each layer.

annotated and labeled at object and pixel levels. For our dataset, we performed the labeling manually. We use these annotations as the ground truth labels. For our manual annotation, we mark the road with the color yellow. We trained the network by using the following tunable hyper-parameters. The parameter values are found based on trial and error with different values and combinations that give the best performance. Some of the hyper-parameters we picked are as follows: learning rate of 2e-4, batch size of 5 and the keep probability of 0.5. We train the network using an NVIDIA RTX2080 Ti GPU; then upload the model to an embedded NVIDIA Xavier computer in our vehicle for real-time inference. From the obtained results of our network we calculate the Cross-Entropy (C.E.) loss. This metric gives a measure of how well our segmentation network is performing. We define our performance goals as C.E. loss, using validation and strive to improve upon another well performing FCN (FCN8). Another important factor for automated driving is performing inference in real-time.

There are many metrics to measure the performance of our segmentation network like mean Intersect over Union (mIoU) and the Cross-Entropy (C.E.) loss. The most common loss function is a pixel-wise Cross-Entropy. In this loss, we compare each pixels predicted class to the ground truth image pixel labels. We repeat this process for all pixels in each image and take the average. In our segmentation we have two classes (i.e. binary), road and not-a-road. The Cross-Entropy loss is defined as:

$$C.E.Loss = -(p.\log(\hat{p}) + (1-p).\log(1-\hat{p})) \tag{1}$$

where the probability of pixel ground-truth values y_{true} for the road defined as $P(Y_{true} = road) = p$ and for not-a-road defined as $P(Y_{true} = notroad) = 1 - p$. The predicted values y_{pred} for road and not-a-road are defined as $P(Y_{pred} = road) = \hat{p}$ and $P(Y_{pred} = notroad) = 1 - \hat{p}$.

4 Obstacle detection and tracking using Kalman filtering

In this section, we use the radar and lidar sensors to detect obstacles and measure their states by processing each sensor data i.e., the radar beat signal and lidar point cloud individually. We then perform a late data fusion or an object-level fusion to add the processed data from the radar and lidar. Then, using a non-linear Kalman Filter method, we take those noisy sensor measurements to estimate and track obstacle states with a higher accuracy than each sensor.

4.1 Radar beat signal data processing

The radar sensor can detect obstacles and their states in a four step process. The first step is to process the noisy digitized mixed or beat signal we receive from the radar. The beat signal is sent through an internal radar ADC to get converted to a digital signal. In the second step, we use a one-dimensional (1D) Fast Fourier Transform (FFT), also known as 1st stage or Range FFT, to transform the signal from the time domain to frequency domain, and separate all its frequency components. The output of the FFT is a frequency response represented by signal power or amplitude in dBm unit versus beat frequency in MHz unit. Each peak in the frequency response represents a detected target. The range FFT output gives us the beat frequency, amplitude, phase, and range (from the equations above) of the targets. To measure the velocity (or Doppler velocity) of the targets, we need to find the Doppler frequency shift, which is the rate of change of phase across radar chirps. The target phase changes from one chirp to another. So, after acquiring the range FFT on the radar chirps, in the third step, we run another FFT (Doppler FFT) to measure the rate of change of phase i.e. the Doppler frequency shift. The output of the Doppler FFT is a 3D map represented by signal power, range, and Doppler velocity. This 3D map is also referred to as Range Doppler Map (RDM). RDM gives us an overview of the targets range and velocity. RDM can be quite noisy since reflected radar signals received can be from unwanted sources like the ground, and buildings, which can create false alarms in our object detection task. In the fourth and final step, these noises or clutters are filtered out to avoid such false positives. One filtering method most used in automotive applications is Cell Averaging Constant False Alarm Rate (CA-CFAR) [10] which is a dynamic thresholding method i.e. it varies thresholds based on the local noise level of the signal. Finally, we have radar CFAR detection with range and Doppler velocity information but object detection and tracking in real-time, is a computationally expensive process. Therefore, we cluster the radar detection that belongs to the same obstacle together to increase the performance of our pipeline. We use a Euclidean distance-based clustering algorithm. In this method, all the radar detection points that are within the size of the target are considered one cluster. The cluster is assigned with a range and velocity at the center equal to the mean of the ranges and velocities of all the cluster detection points.

4.2 Lidar point cloud data processing

Lidar gives us rich information with point clouds (which include position coordinates x, y, z and intensity i) as well as a depth map. The first step to process the lidar point cloud data (PCD) involves downsampling and filtering the data. The raw lidar point cloud is high resolution and covers a long distance (for example a 64-lens laser acquires more than 1 million points per second). Dealing with such a large number of points is very computationally expensive and will affect the real-time performance of our pipeline. Therefore, we downsample and filter. We downsample using a voxel grid; we define a cubic voxel within the PCD and only assign one point cloud per voxel. After downsampling, we use a region of interest (ROI) box to filter any PCD outside of that box that is not of our interest (i.e. points from non-road objects like buildings). In the next step we segment the filtered PCD into obstacles and the road. Knowledge of the road and objects on the road are the two segments most important for the automated driving task. The PCD segmentation happens at the point level; hence, processing would require a lot of resources and slow down the pipeline. In order to improve the performance of our pipeline, similar to the radar data, we cluster the obstacle segments based on their proximity to neighboring points (Euclidean Clustering) and assign each cluster with a new position coordinates (x, y, z) which is the mean of all the point clouds within that cluster. Finally, we can define bounding boxes of the size of clusters and visualize the obstacles with the bounding boxes. For segmenting the PCD into road and obstacles, we need to separate the road plane from the obstacle plane. For this, we use the Random Sample Consensus (RANSAC) method [11].

4.3 Extended Kalman filtering

In the first part of our hybrid framework, we segmented the road scenes at the pixel level into free navigation space for the autonomous vehicle (AV). We also classified the obstacles present that the AV should avoid. In this section, we track those objects, predicting and maintaining their states (2-dimensional positions and velocities) over time. State of the objects being tracked is defined below.

$$\hat{x} = \begin{bmatrix} P_x \\ P_y \\ V_x \\ V_y \end{bmatrix} \tag{2}$$

where P_x and P_y are the object's positions in x and y-direction; V_x and V_y are the object's velocities in x and y-direction. For state tracking, we use Kalman filtering (KF) state estimation method [12]. Our objective is to estimate states that are more accurate and reliable than raw measurements from individual sensors (Lidar or Radar). The motion model for a dynamic system is defined as follows:

$$\hat{x}_{k+1} = F \,.\, \hat{x}_k + B \,.\, \hat{u}_k + \hat{\xi} \tag{3}$$

where $\hat{x}_{k+1}$ is the predicted state of the tracked object, F is the system or state transition matrix, $\hat{x}$ is the state vector, B is the control input matrix, $\hat{u}$ is the input vector, and ξ is the process or motion model noise. We assume a linear motion model in which the object travels at a constant velocity for our test, but in reality, the object may not maintain a constant velocity. In object tracking, we do not know the exact movement of the tracked object; hence, we will not be able to model the dynamics of the object so we assume no inputs i.e. $B.\hat{u} = 0$. Instead, we subsume the motion uncertainty into the process noise ξ. We also observe the system by a measurement or observation model that maps the state of the object into the measurement space of our sensors (lidar or radar). The measurement model is as follows:

$$\hat{z}_{k+1} = H \,.\, \hat{x}_{k+1} + \hat{\epsilon} \tag{4}$$

where $\hat{z}_{k+1}$ is the sensor measurement vector, H is the measurement function, $\hat{x}$ is the state vector, $\hat{\epsilon}$ is the sensor measurement noise. A KF involves three steps: initialization, prediction, and update. For the predict step, we use the objects current state (2D-position and 2D-velocity) to estimate the state at the next time step using eqn. 3. Substituting the state transition matrix F and the measurement noise $\hat{\xi}$ definitions into eqn. 3; the motion model becomes:

$$\begin{bmatrix} P_{x_{k+1}} \\ P_{y_{k+1}} \\ V_{x_{k+1}} \\ V_{y_{k+1}} \end{bmatrix} = \begin{bmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} P_{x_k} \\ P_{y_k} \\ V_{x_k} \\ V_{y_k} \end{bmatrix} + \begin{bmatrix} \xi_{px_k} \\ \xi_{py_k} \\ \xi_{vx_k} \\ \xi_{vy_k} \end{bmatrix} \tag{5}$$

In cases where the object did not maintain the same velocity, we represent the motion uncertainty in the predicted covariance equation. Predicted covariance is calculated from:

$$P_{k+1} = F \,.\, P_k \,.\, F^T + Q \tag{6}$$

where P_{k+1} is the uncertainty associated with our predictions. In the update step, we calculate the measurement residual by comparing the measured sensor readings from the actual perfect world sensor readings:

$$\hat{y}_{k+1} = \hat{z}_{k+1} - H \,.\, \hat{x}_{k+1} \tag{7}$$

also, we calculate the residual covariance from:

$$\hat{s}_{k+1} = H \,.\, P_{k+1} \,.\, H^T + R \tag{8}$$

Moreover, we have K, the Kalman gain, which combines the uncertainty of our predicted state P_{k+1} with the uncertainty of sensor measurements S_{k+1}.

$$K = P_{k+1} \,.\, H^T \,.\, {S_{k+1}}^{-1} \tag{9}$$

At last, we calculate the updated state estimate as well as its covariance:

$$\hat{x}_{k+1} = \hat{x}_k + K \,.\, \hat{y} \tag{10}$$

$$P_{k+1} = (I - K \,.\, H) \,.\, P_k \tag{11}$$

The standard Kalman filter can only be used when our models are linear (motion model or the measurement model.) with the assumption that our data has a Gaussian distribution. However, in a system, the motion model or measurement model or both can be nonlinear; so, the standard KF may not converge, and its equations are not valid. In this case, to work with nonlinear models, we use the classic Extended Kalman filter (EKF). EKF fixes this problem by linearizing the nonlinear functions (state transition or measurement functions) around the mean of the current state estimate using Taylor series expansion and taking the Jacobian of our functions [13]. In EKF, the system equations are:

$$\hat{x}_{k+1} = f(\hat{x}_k, \hat{u}_k) + \hat{\xi} \tag{12}$$

$$\hat{z}_{k+1} = h(\hat{x}_{k+1}) + \hat{\epsilon} \tag{13}$$

where $f(\hat{x}_k, \hat{u}_k)$ and $h(\hat{x}_{k+1})$ are the nonlinear state transition and measurement functions respectively. We linearize these functions by finding their Jacobians, so the linearized system becomes:

$$\hat{x}_{k+1} = F \,.\, \hat{x}_k + \hat{\xi} \tag{14}$$

$$\hat{z}_{k+1} = H \,.\, \hat{x}_{k+1} + \hat{\epsilon} \tag{15}$$

where F and H are approximated as: $F = \frac{\partial f}{\partial x}$ and $H = \frac{\partial h}{\partial x}$. For our tracking problem, we take advantage of two sensor measurements: lidar and radar. the ground truth (the actual path of the object) is measured manually and used for calculating root mean square error (RMSE) from eqn. 16 to measure our object tracking algorithm performance.

$$R.M.S.E. = \sqrt{\frac{1}{n} \cdot \sum_{i=1}^{n} {||\hat{x}_{estimate} - \hat{x}_{groundtruth}||_2}^2} \tag{16}$$

5 Experimental results

The results of each part of our hybrid sensor fusion framework are presented in this section. For each part, we measure their performance using an evaluation metric appropriate for that method; for EKF fusion and object tracking, we use the root mean square error (RMSE) metric, and for FCN road segmentation we use the Cross-Entropy (CE) loss metric. First, we show the results of our FCNx algorithm in fig. 3. In all images shown, by visual examination we can see that our model (row three) performs the road segmentation better and closer to the ground truth (row two) than the benchmark FCN8 (row four). The FCN8 segmentation was not as smooth as our model at segmenting road border lines. For example, it classified parts of the right-side parking area and right-side sidewalk as the road in the row four columns three and four images, respectively. It also showed some difficulty with False Positives (FP). For example, it falsely classified parts of the left-side grass area and parts of the bicycle path on the right side as the road in the row four column one and two images, respectively.

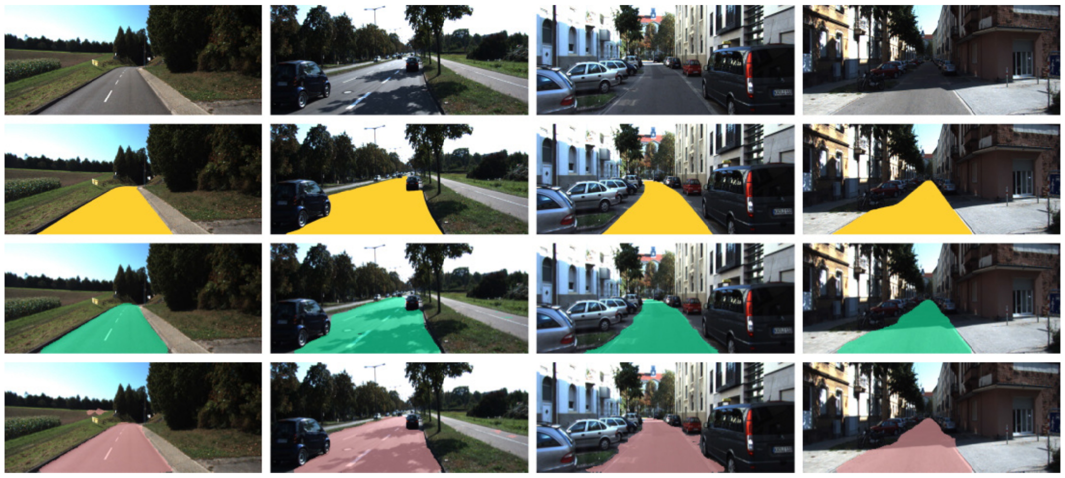
Figure 3: Results of semantic segmentation of the road. First row: some examples of our input images in various situations. From left to right: Highway clear weather no traffic, highway shadow mix with traffic, urban dim with cars parked, and urban shadow mix with cars parked. Second row: the corresponding road ground truth annotations in yellow overlaid over the original image. Third row: road segmentation by our model FCNx. Row four: road segmentation by UC Berkeley FCN8.

Also analytically, we compare our proposed network performance with FCN8 network performance by measuring the Cross-Entropy loss and inference time metrics. Under the same conditions, i.e. the same training input data and hyperparameter choices; our model FCNx reduces the CE loss by 3.2 % compared to FCN8 model while maintaining similar inference time.

Table 1: Comparison of our architecture performance with FCN8 network.

Method	Cross Entropy Loss %	Inference Time (ms)
FCN8	6.8	∼180
FCNx (Our model)	3.6	∼185

Results of the Extended Kalman filter estimation for a real-world example is shown in fig. 4. The red line shows the actual path of the vehicle we are tracking and the blue and yellow points are the lidar and radar measurements, respectively. The orange points are the EKF fused output. The EKF fused predictions are closer to the actual path than individual lidar and radar measurements.

We evaluate the performance of our Extended Kalman filter predictions by measuring the root mean square error. RMSE measures how well our prediction values fit the actual path of the target vehicle. In our experiment, we achieve RMSE of 0.065 and 0.061 for the x and y-position of our tracked target vehicle, which shows an improvement over individual lidar and radar sensors.

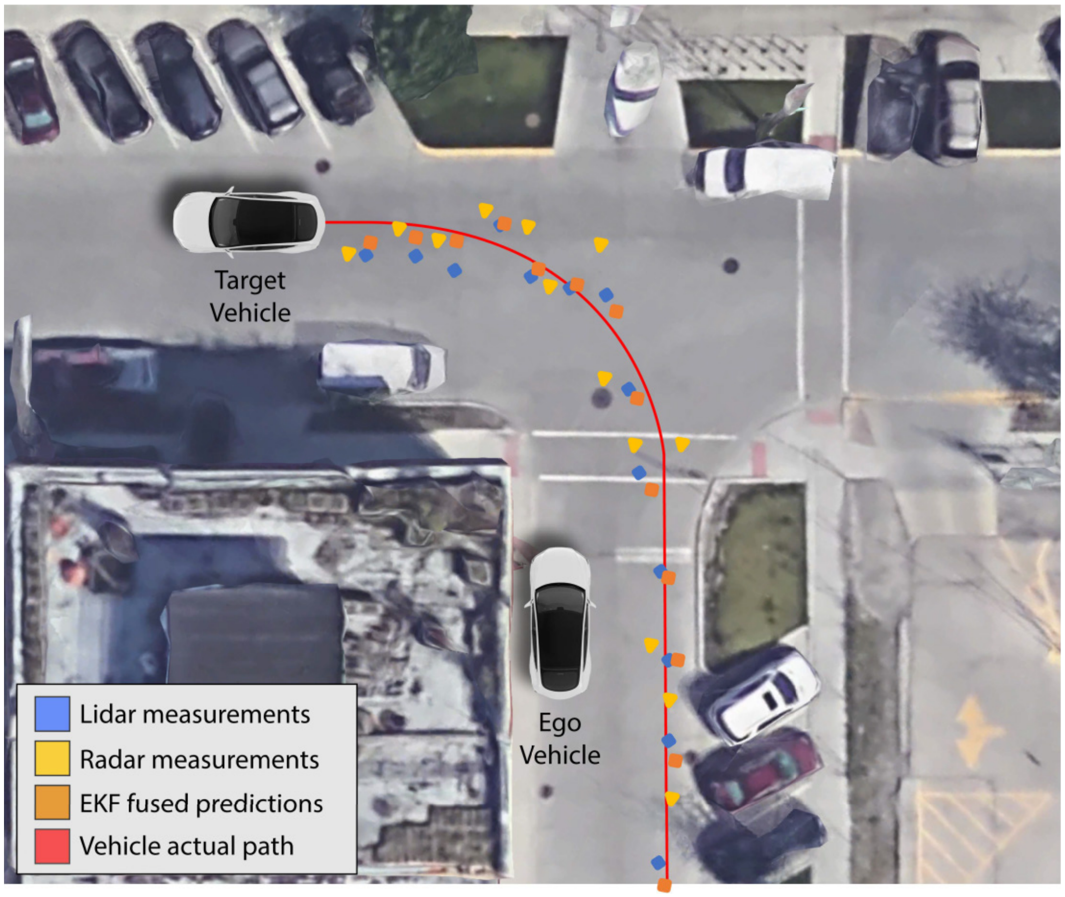

Figure 4: Results of EKF state predictions of a target vehicle from sensors on an ego vehicle.

6 Conclusion

The results of each fusion method in our hybrid sensor fusion algorithm gives our AV a detailed map of the environment. Having a clear understanding of the surrounding environment can result in an optimal decision making, and generating optimal control inputs to the actuators (accelerator, brakes, steering) of our AV. In this paper, a new sensor fusion framework configuration is proposed and successfully demonstrated real-time environment perception in a vehicle. Appropriate combinations of Vision + Lidar and Vision + Radar sensor fusion is successfully demonstrated, using a hybrid pipeline: deep learning and EKF. The algorithm uses our proposed FCNx deep learning framework and is implemented in edge-computing device in real-time. For future work, cloud computing and edge computing coordination would be the next step to further enhance this framework.

References

[1] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[2] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.

[3] Hyeonwoo Noh, Seunghoon Hong, and Bohyung Han. Learning deconvolution network for semantic segmentation. *The IEEE International Conference on Computer Vision (ICCV)*, December 2015.

[4] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[5] Mohammad Javad Shafiee, Brendan Chywl, Francis Li, and Alexander Wong. Fast yolo: a fast you only look once system for real-time embedded object detection in video. *arXiv preprint arXiv:1709.05943*, 2017.

[6] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

[7] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. *European conference on computer vision*, pages 21–37, 2016.

[8] Fisher Yu, Wenqi Xian, Yingying Chen, Fangchen Liu, Mike Liao, Vashisht Madhavan, and Trevor Darrell. Bdd100k: A diverse driving video database with scalable annotation tooling. *arXiv preprint arXiv:1805.04687*, 2018.

[9] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. *The International Journal of Robotics Research*, 32(11):1231–1237, 2013.

[10] Chr Kabakchiev, Lyubka Doukovska, and Ivan Garvanov. Cell averaging constant false alarm rate detector with hough transform in randomly arriving impulse interference. *Cybernetics and Information Technologies*, 6(1):83–89, 2006.

[11] Martin A. Fischler and Robert C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, June 1981.

[12] Greg Welch, Gary Bishop, et al. An introduction to the kalman filter. 1995.

[13] Lennart Ljung. Asymptotic behavior of the extended kalman filter as a parameter estimator for linear systems. *IEEE Transactions on Automatic Control*, 24(1):36–50, 1979.